STORM: Lightning-Fast Resource Management

Eitan Frachtenberg, **Fabrizio Petrini**, Juan Fernandez, Scott Pakin, and Salvador Coll
fabrizio@lanl.gov

http://www.c3.lanl.gov/~fabrizio

Performance and Architecture Laboratory

CCS-3 Modeling, Algorithms, and Informatics Group

Los Alamos National Laboratory

SC2002

Grand Vision

- More effective use of cluster resources
  - Lower response time
  - Higher throughput
- *Transparent* fault tolerance
  - No application modifications

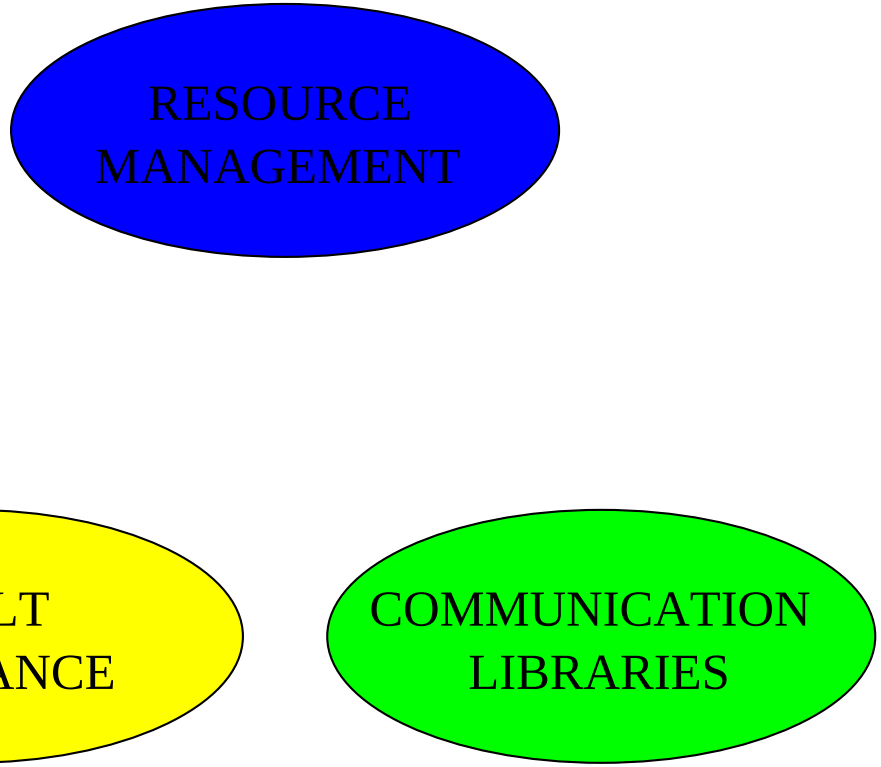
Buffered Coscheduling

Buffered Coscheduling (BCS) is a new methodology to:

- improve system responsiveness
- increase resource utilization,
- tolerate inefficient programs (with communication and load imbalance),
- implement fault-tolerance,

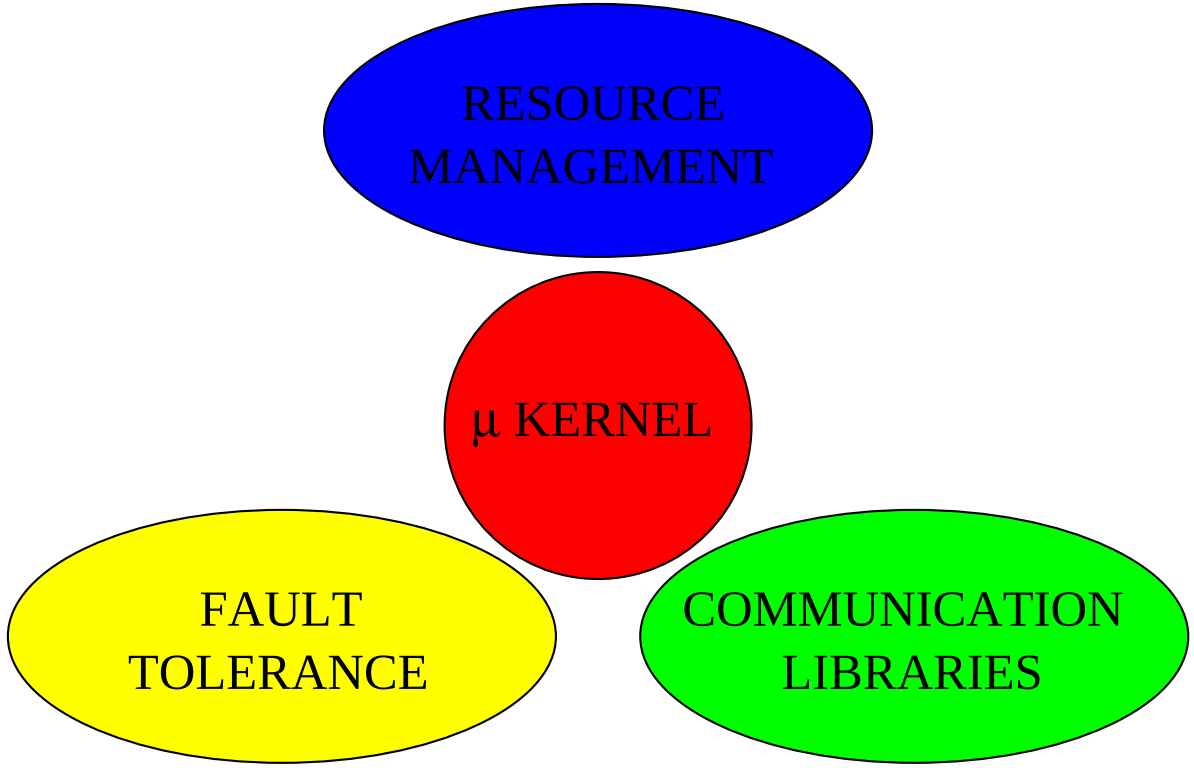
in a large-scale parallel computer or a cluster of workstations

Buffered Coscheduling: Vision



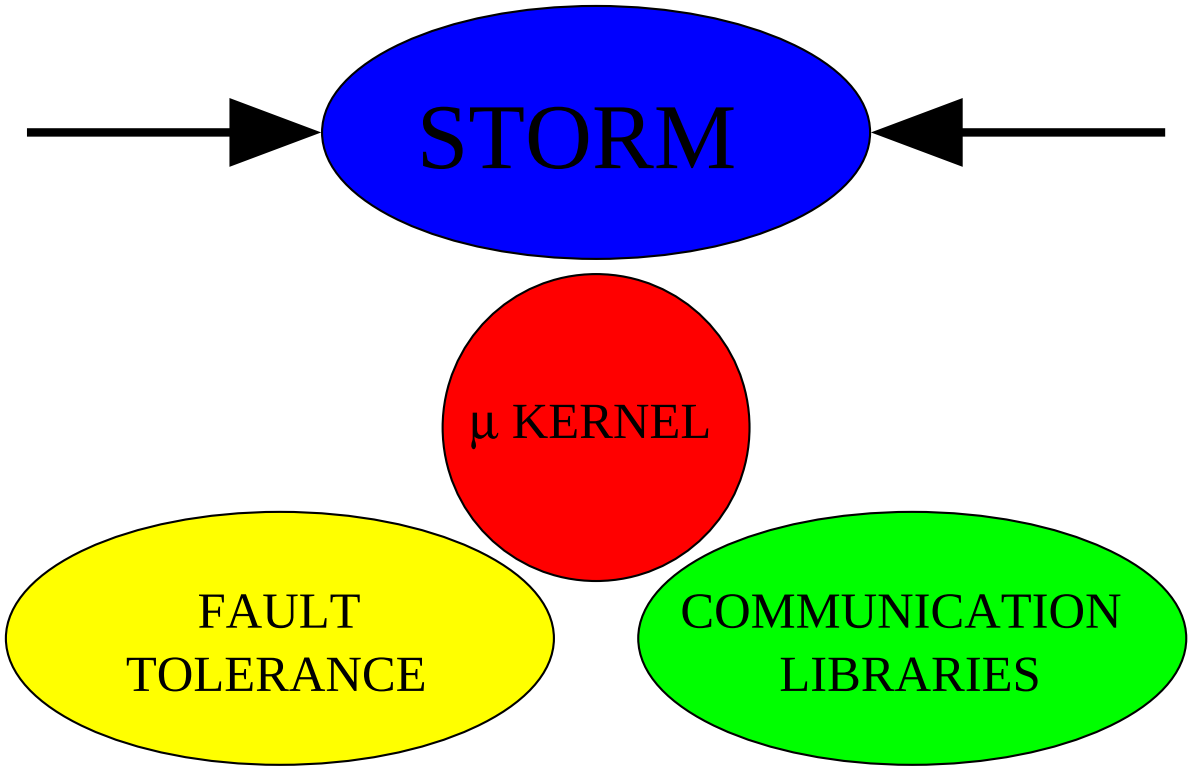
Buffered Coscheduling tries to achieve these goals by greatly simplyfing the system software (resource management, communication libraries and fault-tolerance)

Buffered Coscheduling: Implementation



Buffered Coscheduling implements resource management, communication libraries and fault-tolerance on top of a common μkernel

STORM



In this talk we will focus on STORM, a resource manager implemented on the Buffered Coscheduling μkernel

Resource Management

- Resource allocation for parallel jobs
- Job launch and termination
- Cluster management
- Monitoring and Debugging

STORM

- STORM (Scalable TOol for Resource Management) uses the Buffered Coscheduling μkernel
- Easy to port
- The key innovation behind STORM is a software architecture that enables resource management to exploit low-level network features
- As a result of this HPC-application-like design, STORM is orders of magnitude faster than the best reported results in the literature

Outline

- Overview of resource management
- STORM architecture
- Implementation
- Performance evalution
- Scalability analysis

Characteristics of Desktop versus Cluster

Characteristic	Desktop	Cluster
Mean time between user-visible failures	Years	Days down to hours
Scheduling	Timeshared	Batch queued or gang scheduled with large quanta
Job-launching speed	< 1 second	Arbitrarily long (batch) or many seconds (gang scheduled)

State of the art in Resource Management

Resource Managers (e.g., PBS, LSF, RMS, LoadLeveler, Maui) are typically implemented using

- TCP/IP – favors portability over performance
- Non scalable algorithms for the distribution/collection of data and control messages – favors development time over performance
- Performance non important for small clusters, but crucial for large clusters $\rightarrow$ need fast and scalable resource management

State of the art in Resource Management

Resource Managers (e.g., PBS, LSF, RMS, LoadLeveler, Maui) are typically implemented using

- TCP/IP – favors portability over performance
- Non scalable algorithms for the distribution/collection of data and control messages – favors development time over performance
- Performance non important for small clusters, but crucial for large clusters $\rightarrow$ need fast and scalable resource management

If the cluster has a powerful, scalable network, why aren't we using it?

STORM mechanisms

STORM is based on only three mechanisms

XFER-AND-SIGNAL Transfer (PUT) a block of data from local memory to the global memory of a set of nodes (possibly a single node).

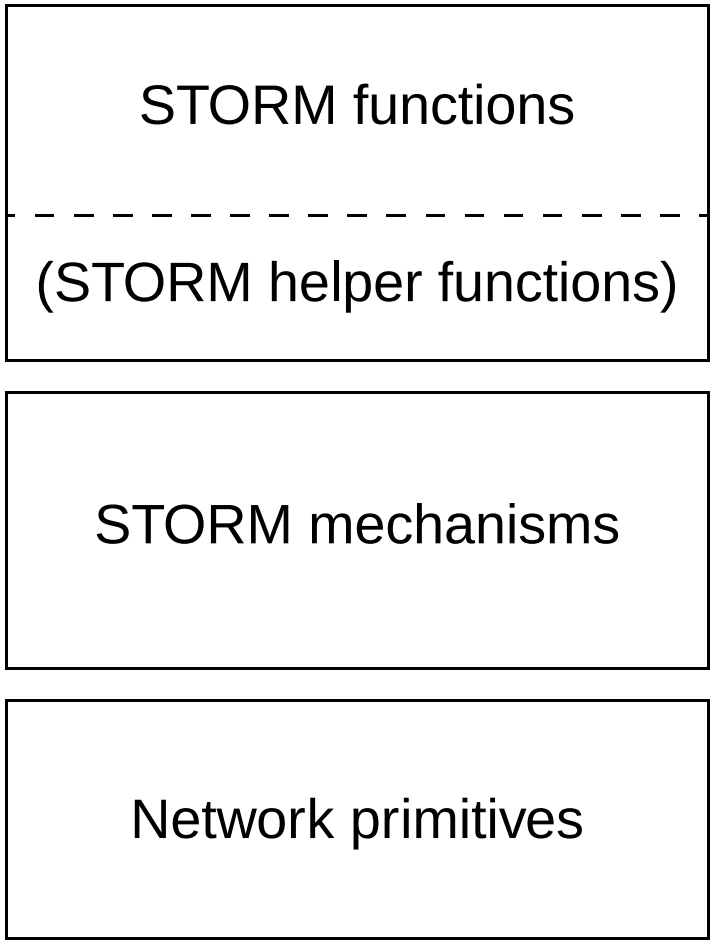
TEST-EVENT Local synchronization

COMPARE-AND-WRITE Global query with reduction variable.

Efficient and scalable implementation of these mechanisms

$\rightarrow$ STORM scalable

STORM implementation structure



heartbeat, file transfer, termination detection

flow control, queue management

XFER-AND-SIGNAL, TEST-EVENT, COMPARE-AND-WRITE

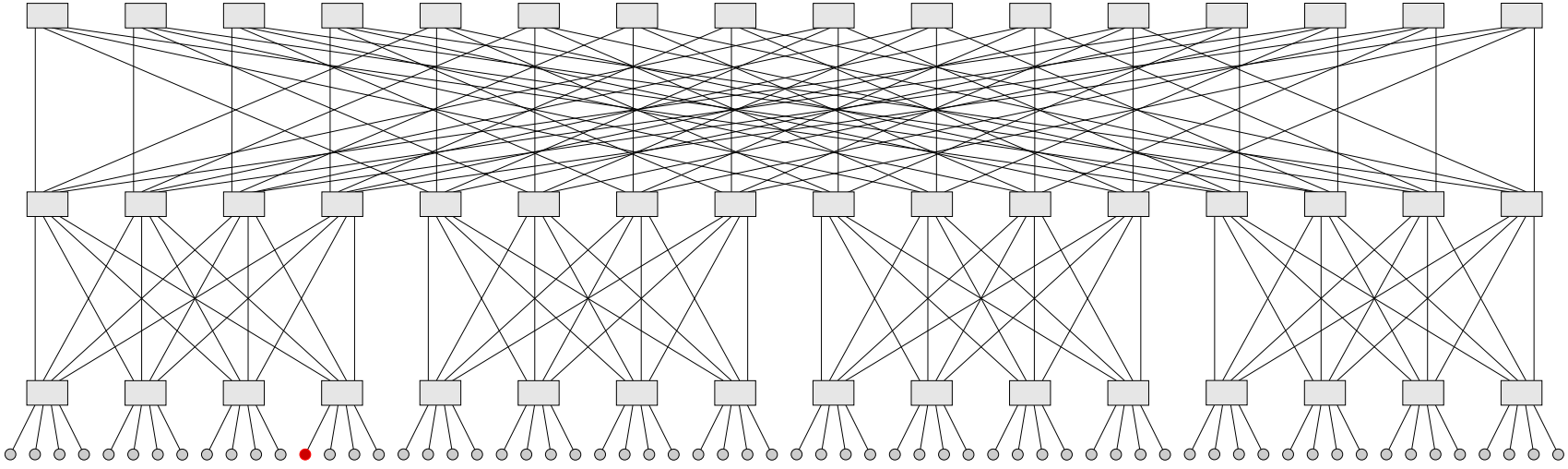
remote DMA, network conditionals, event signaling, . . .

Portability of the STORM mechanisms

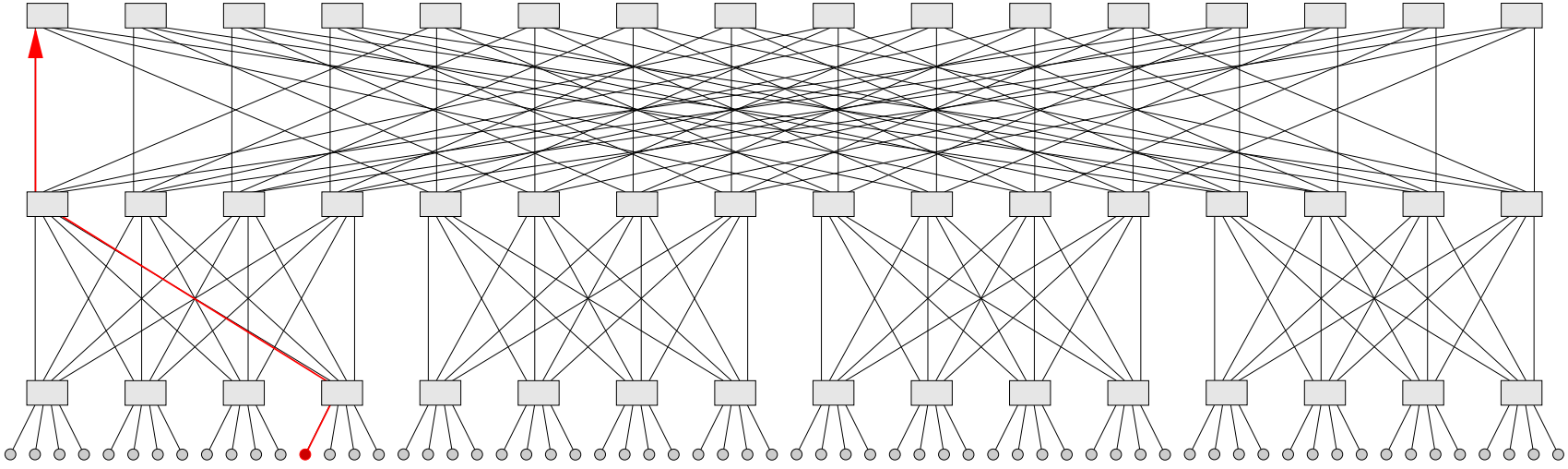
Network	COMPARE-AND-WRITE (μS)	XFER-AND-SIGNAL (MB/s)
Gigabit Ethernet	$46 \log n$	Unknown
Myrinet	$20 \log n$	$\sim 15n$
Infiniband	$20 \log n$	Unknown
QsNET	$\mathbf{< 10}$	$\mathbf{> 150n}$
BlueGene/L	< 2	$700n$

SC2002

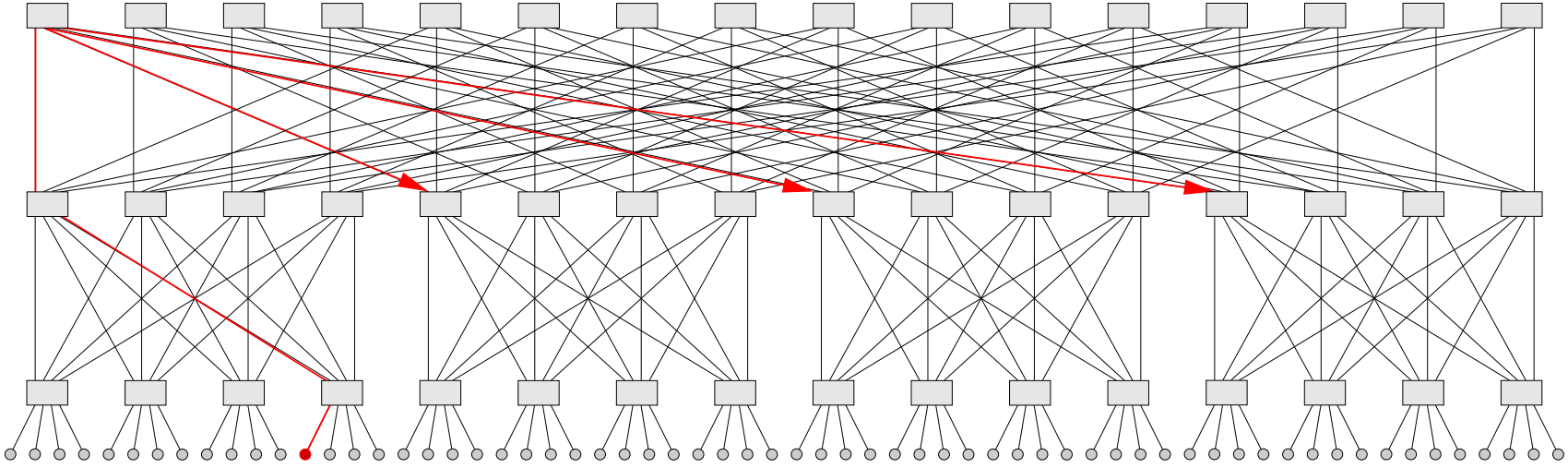
Hardware support for XFER-AND-SIGNAL



Hardware support for XFER-AND-SIGNAL

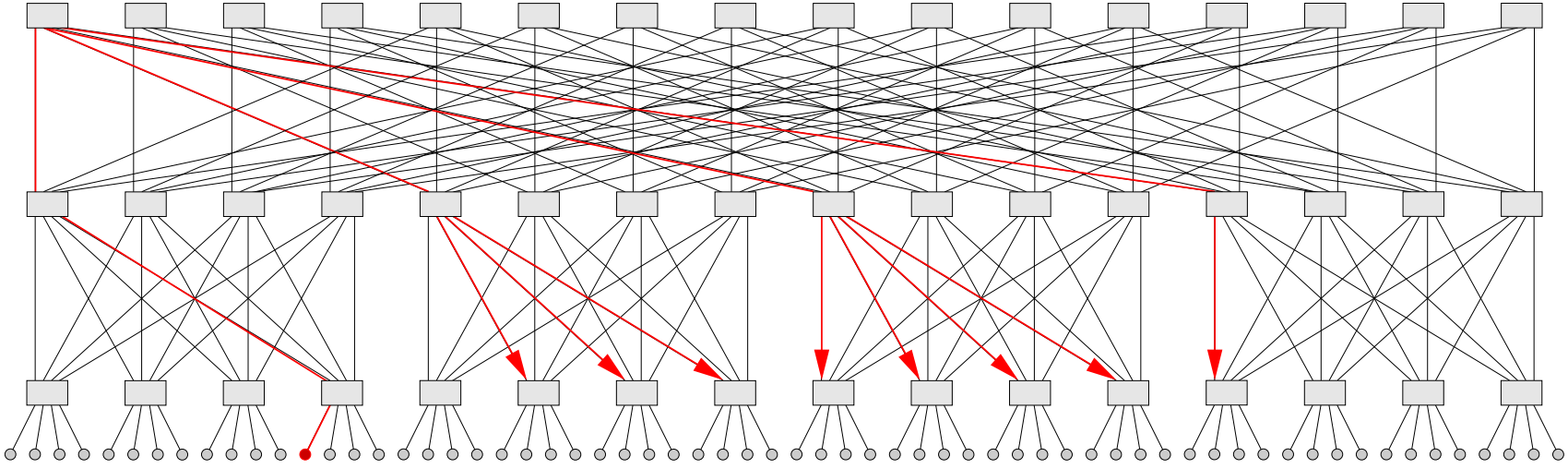


Hardware support for XFER-AND-SIGNAL

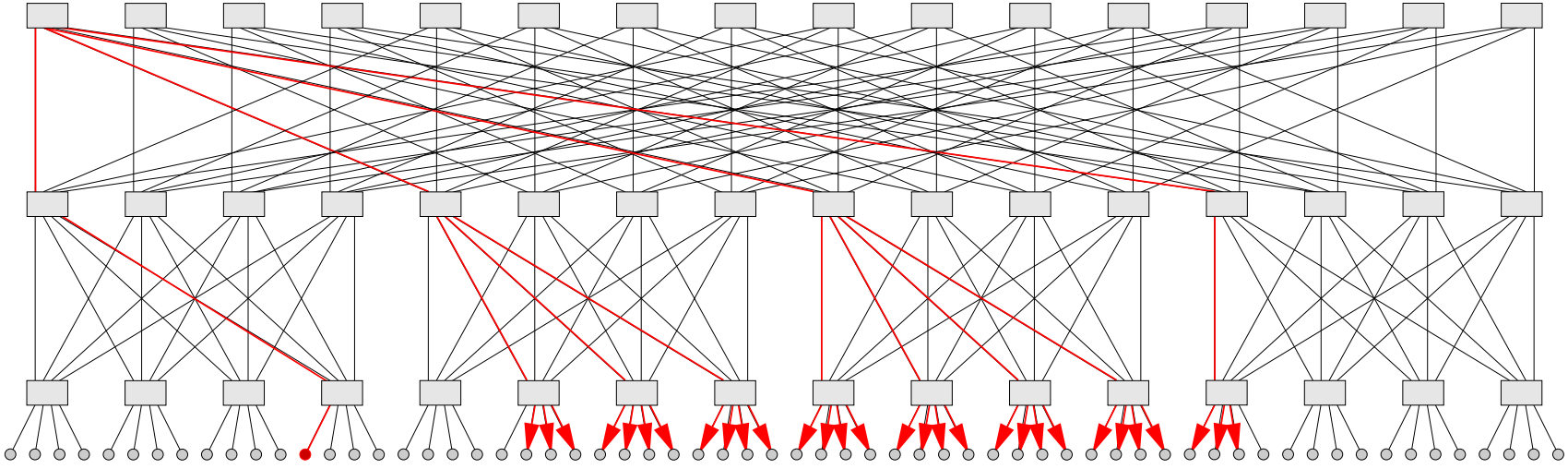


SC2002

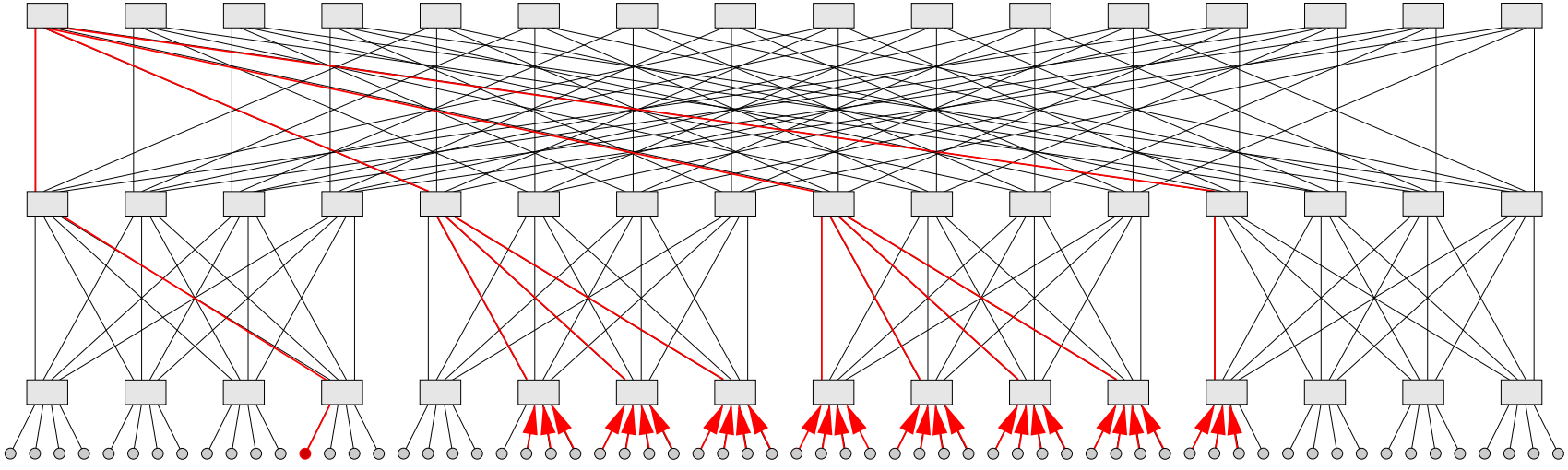
Hardware support for XFER-AND-SIGNAL



Hardware support for XFER-AND-SIGNAL

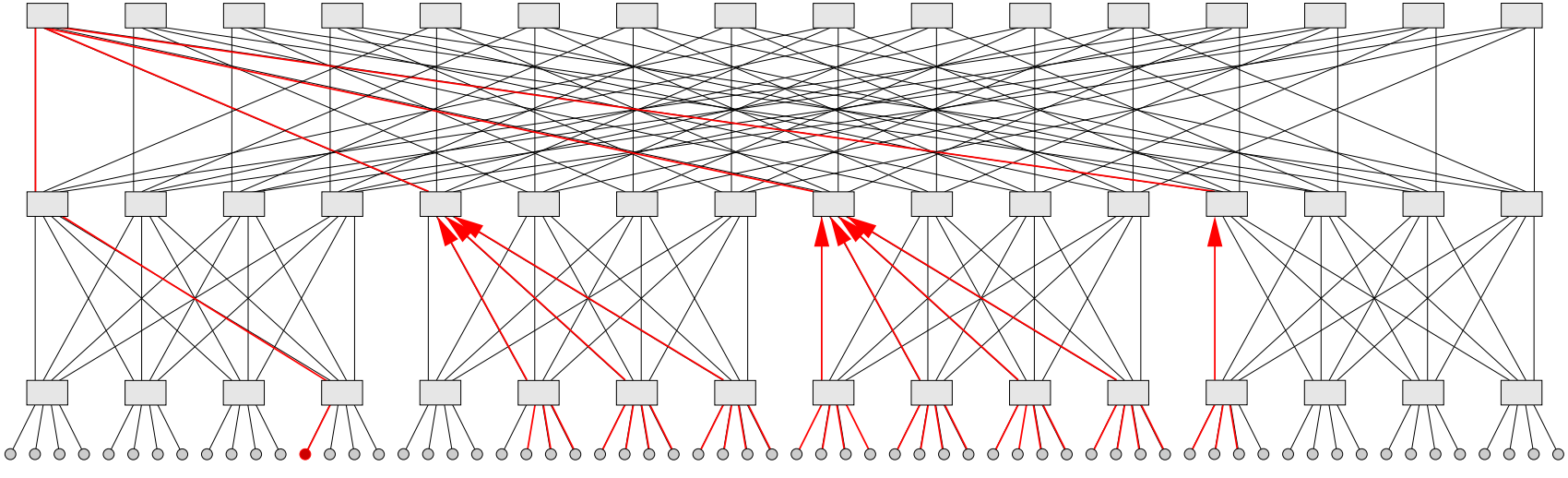


rdware support for COMPARE-AND-WR



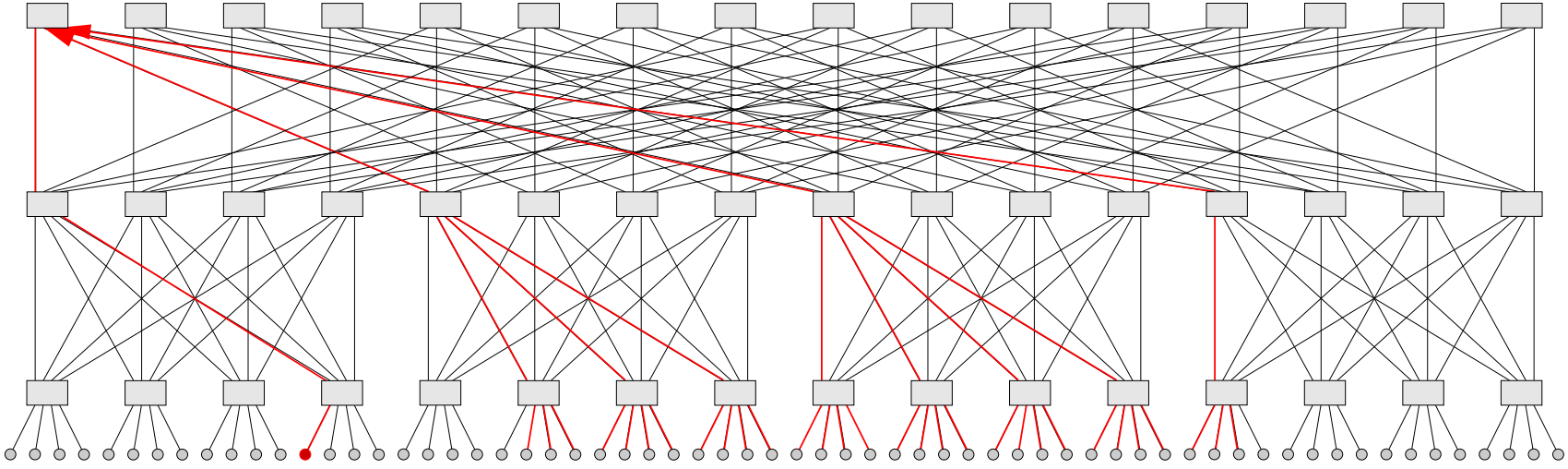
SC2002

rdware support for COMPARE-AND-WR

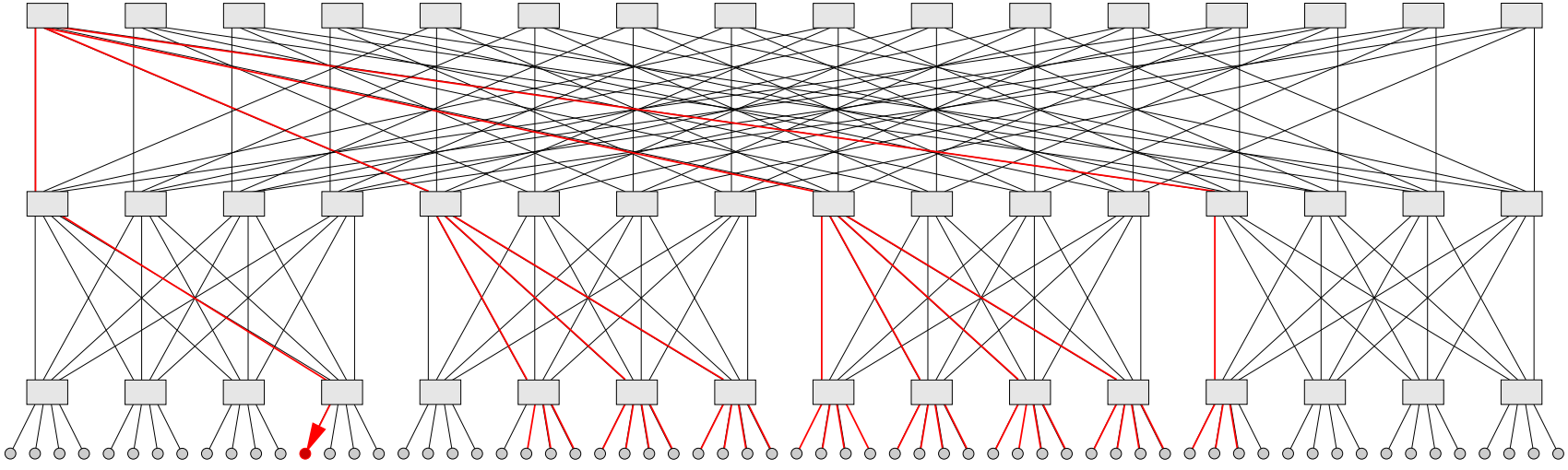


SC2002

rdware support for COMPARE-AND-WR

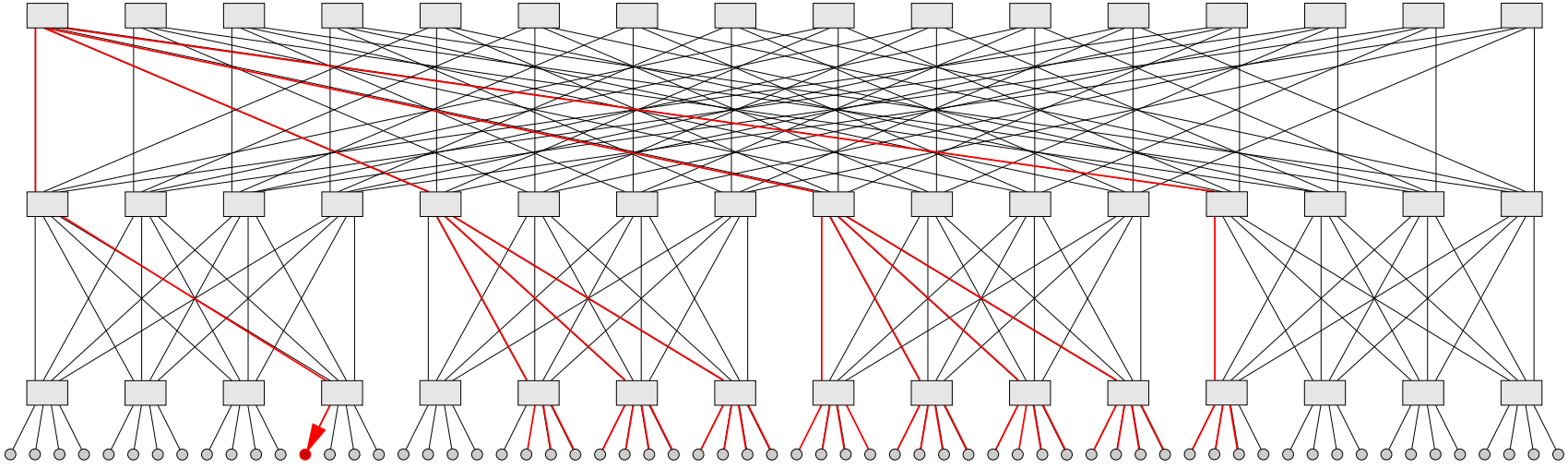


rdware support for COMPARE-AND-WR



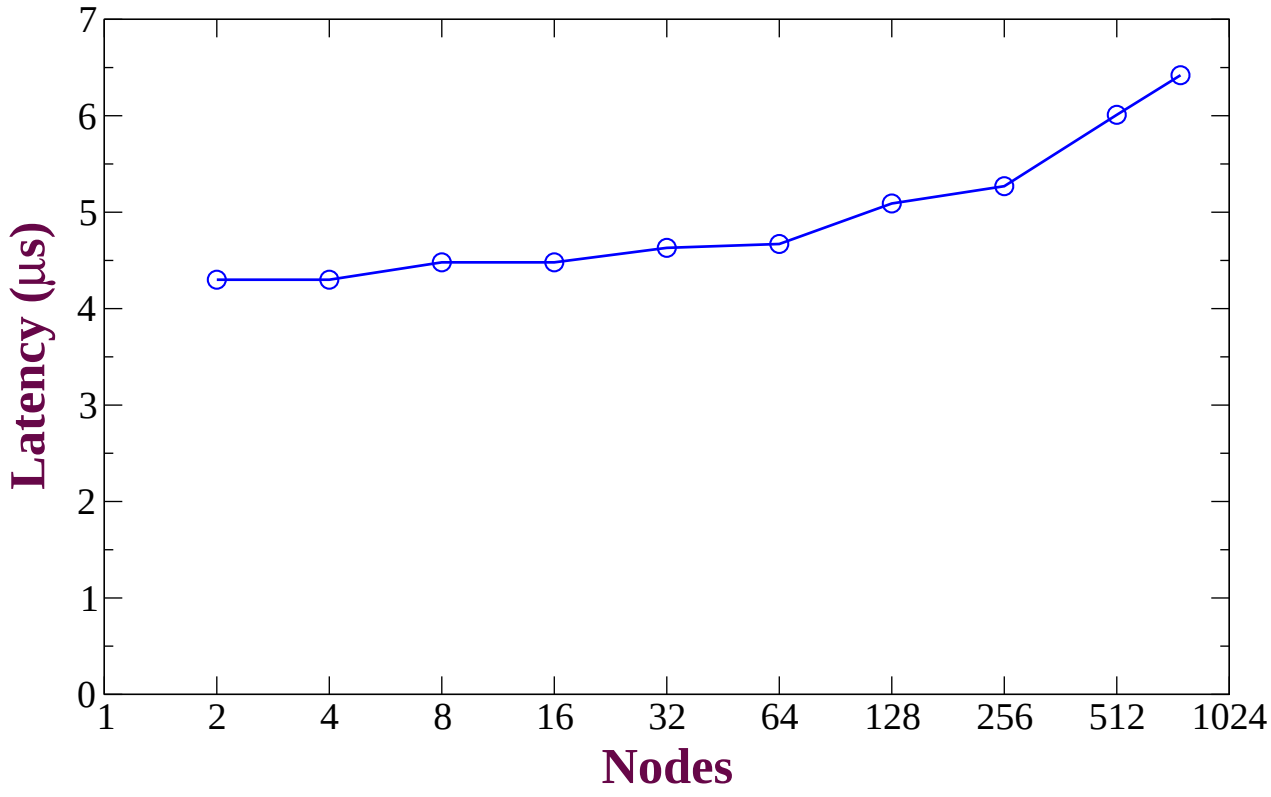
SC2002

rdware support for COMPARE-AND-WR



The STORM mechanisms XFER-AND-SIGNAL and COMPARE-AND-WRITE can be easily and efficiently implemented on top of the hardware broadcast.

Scalability of the STORM Mechanisms

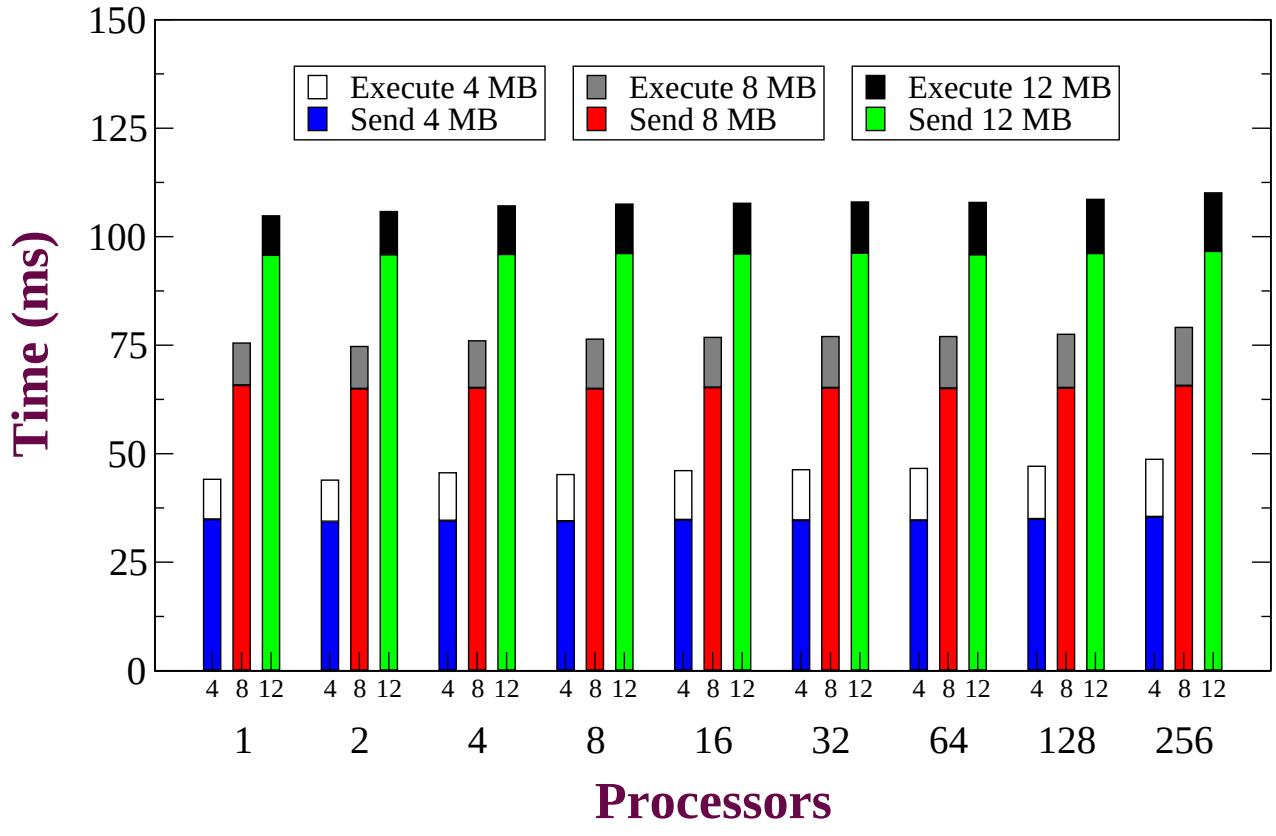


XFER-AND-SIGNAL and COMPARE-AND-WRITE scale efficiently on Lemieux, Pittsburgh Supercomputing Center. Less than 10 μs on 768 nodes/3072 processors

Experimental Results

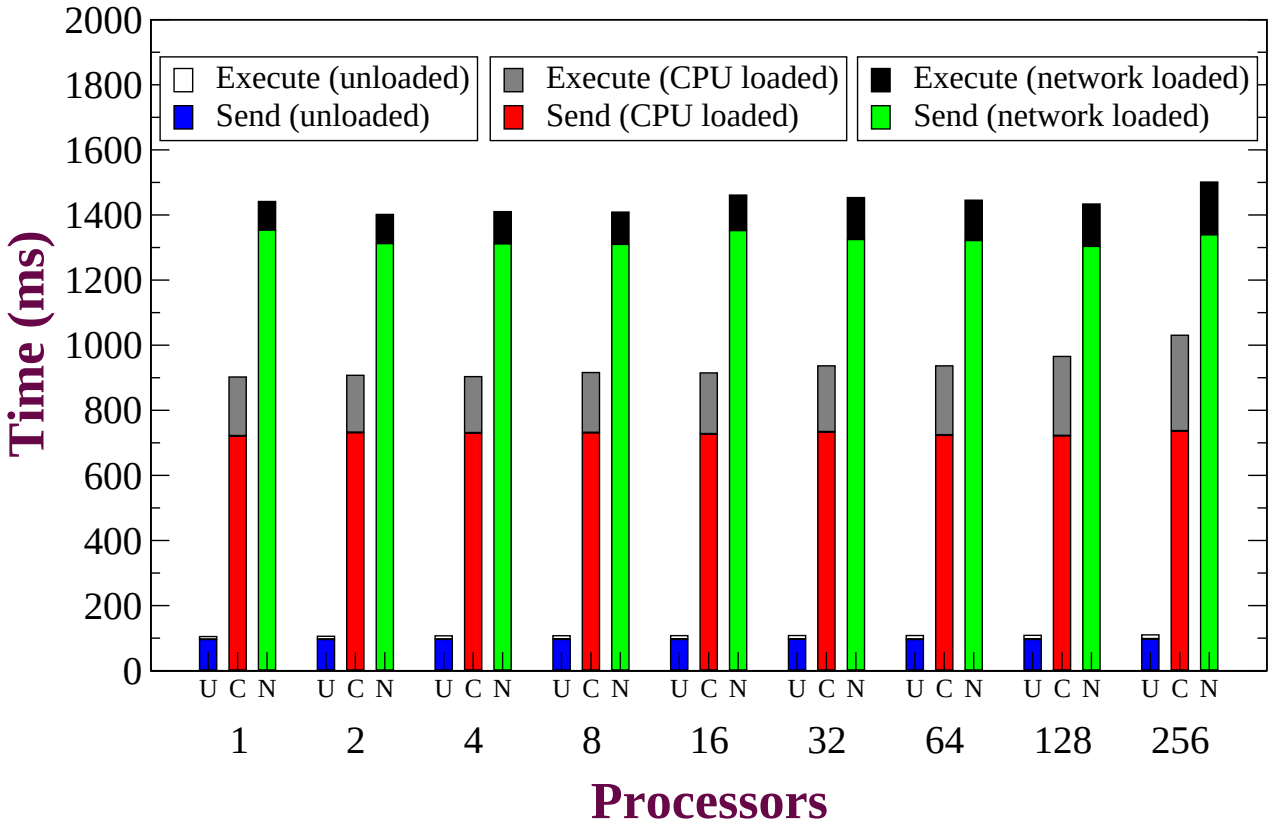
- 64 nodes/256 processors ES40 Alphaserver cluster
- 2 independent rails of Quadrics
- Linux 2.4.3
- Files are placed in ramdisk, in order to avoid I/O bottlenecks and expose the performance of the resource management algorithms.

Launch times (unloaded system)



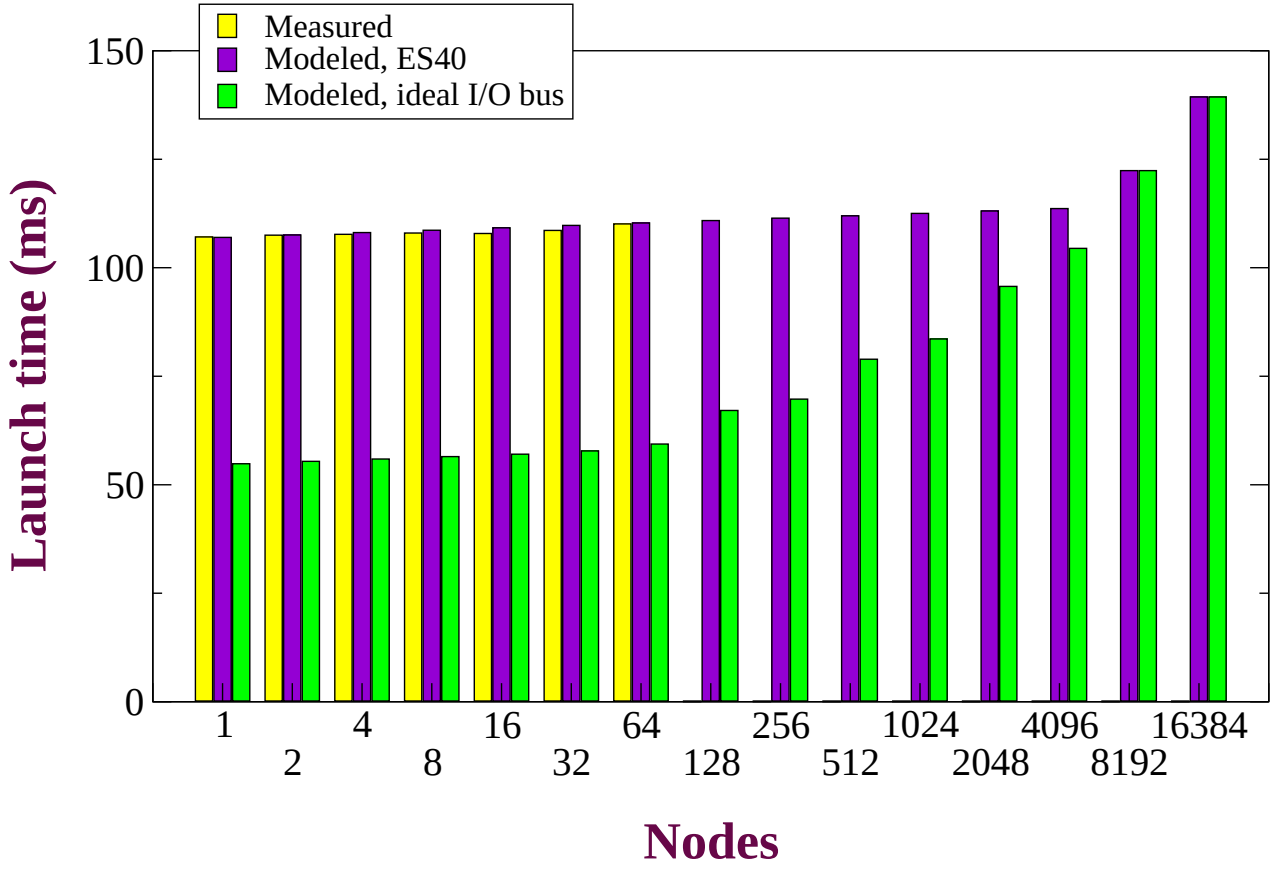
The launch time is constant when we increase the number of processors. STORM is highly scalable

Launch times (loaded system 12MB)



- Launch time is more sensitive to network load rather than CPU load
- In the worst-case scenario it still takes only 1.5 seconds to launch a 12 MB file on 256 processors

Measured and estimated launch times



The model shows that in an ES40-based Alphaserver a 12 MB binary can be launched in 135 ms on 16,384 nodes

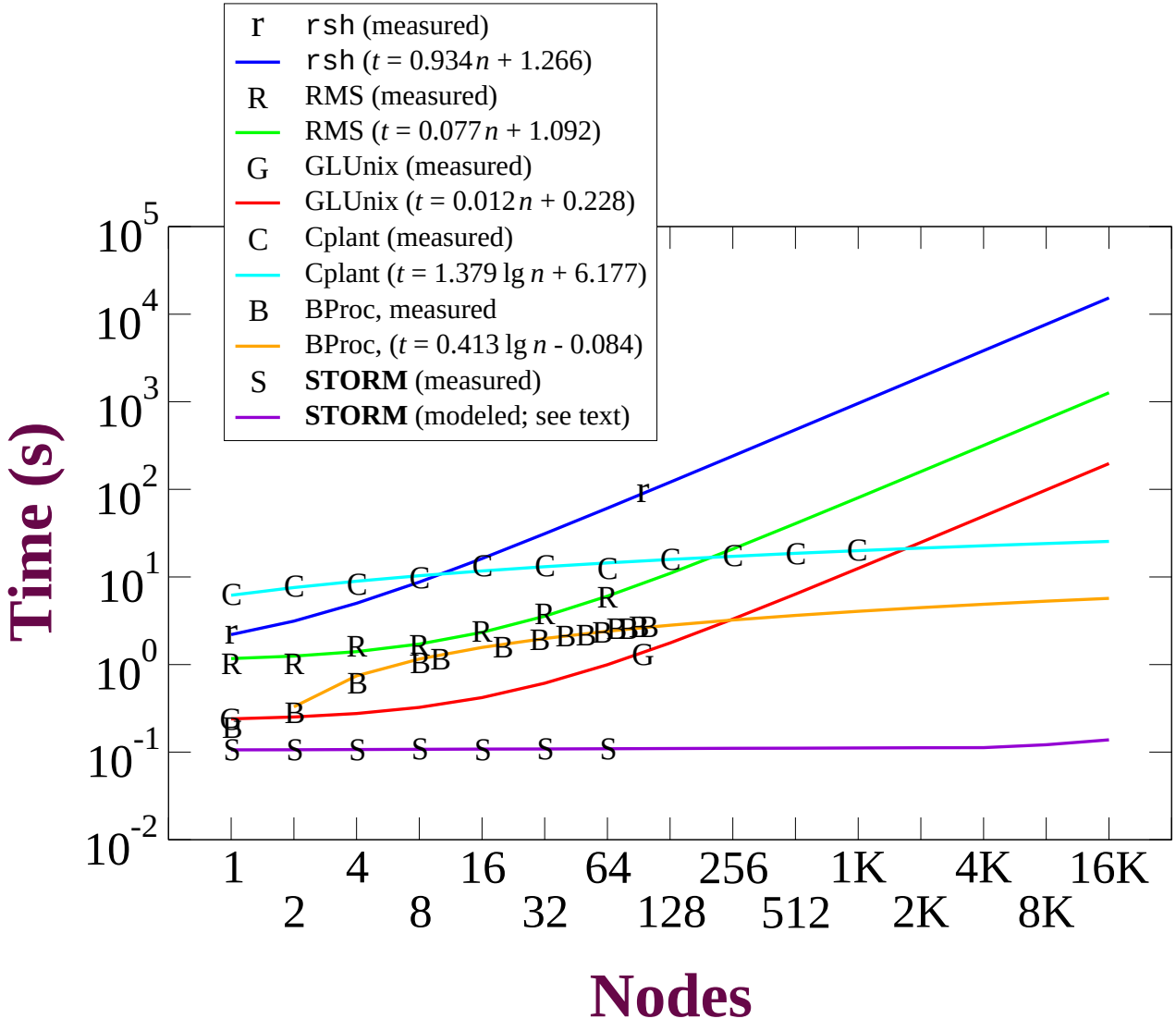
SC2002

Measured and predicted performance of existing job launchers

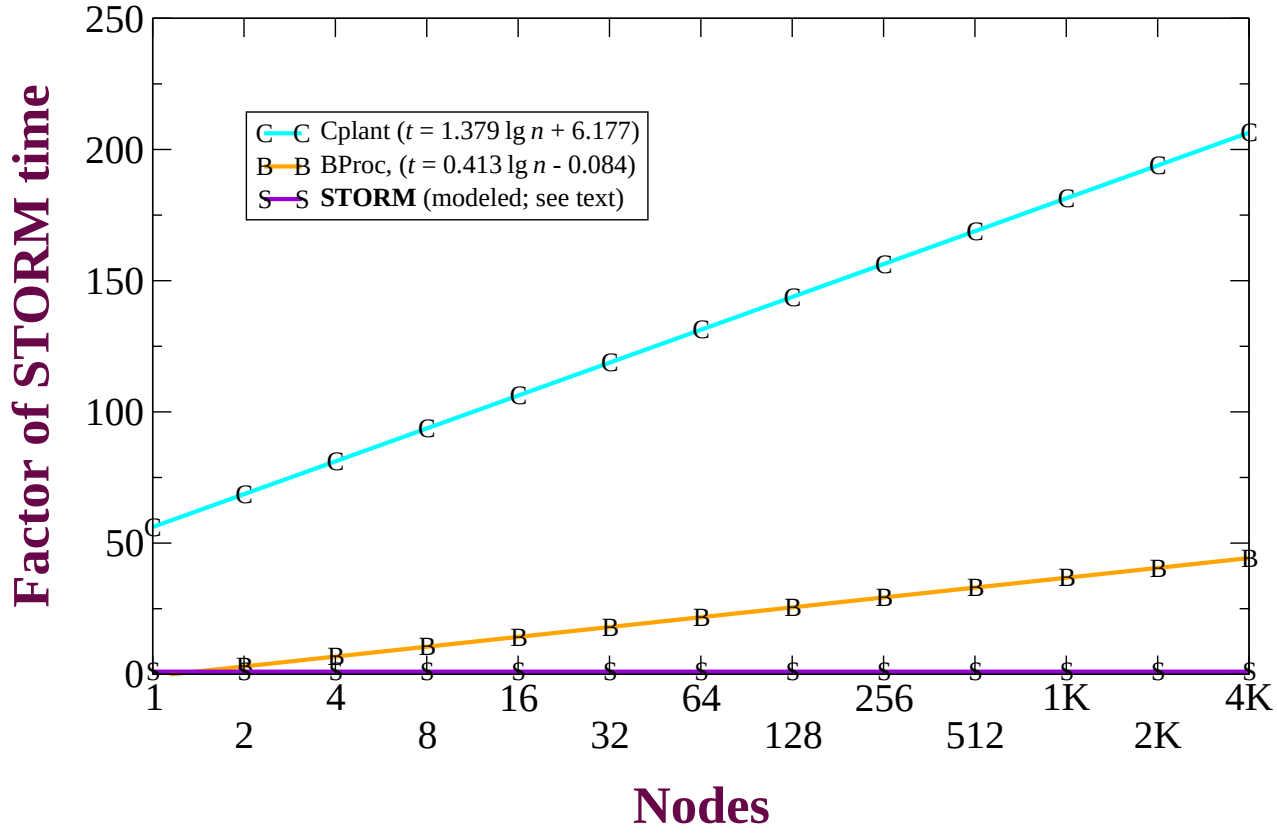
We compare the job launching performance of STORM with

- rsh (remote shell)
- RMS (Resource Management System), production resource manager of the Quadrics network
- Sandia's Cplant
- Bproc

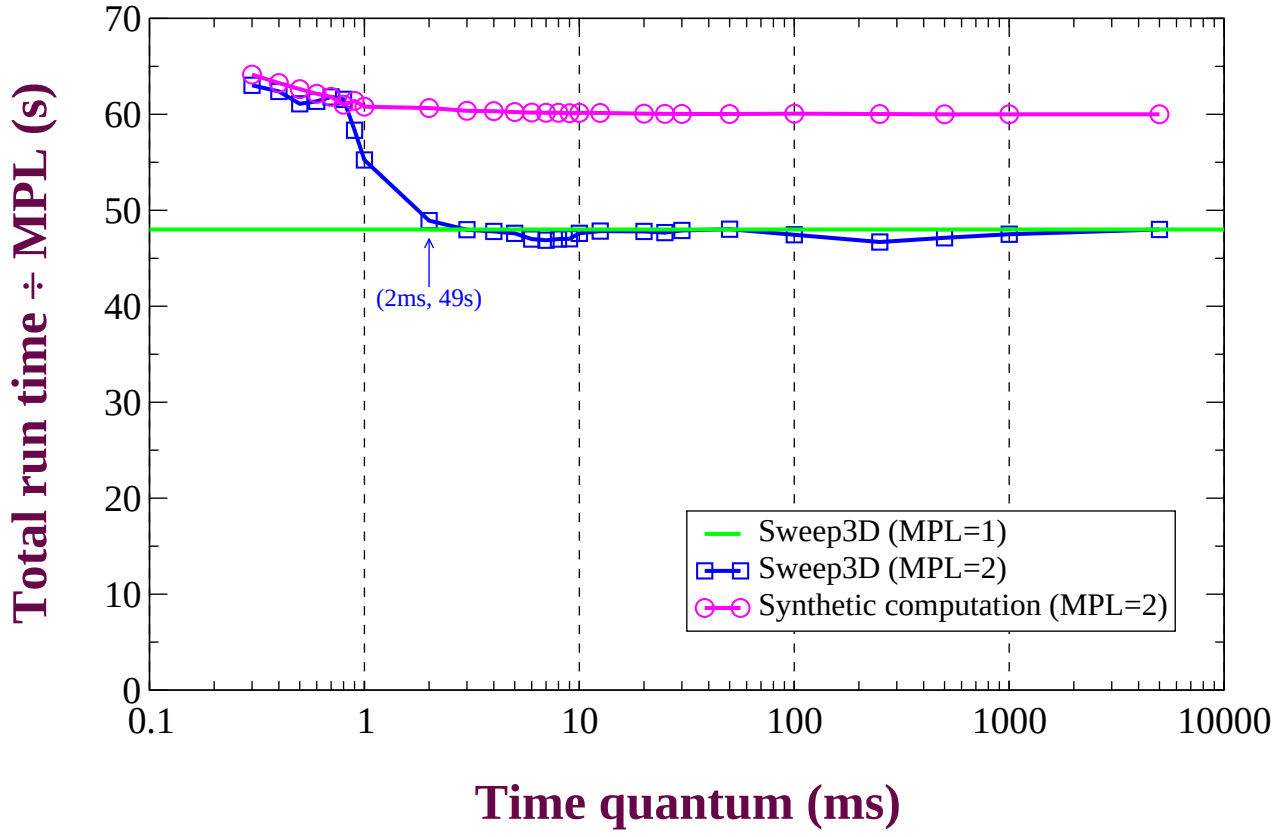
Measured and predicted performance of existing job launchers



Relative performance of Cplant, BProc, and STORM

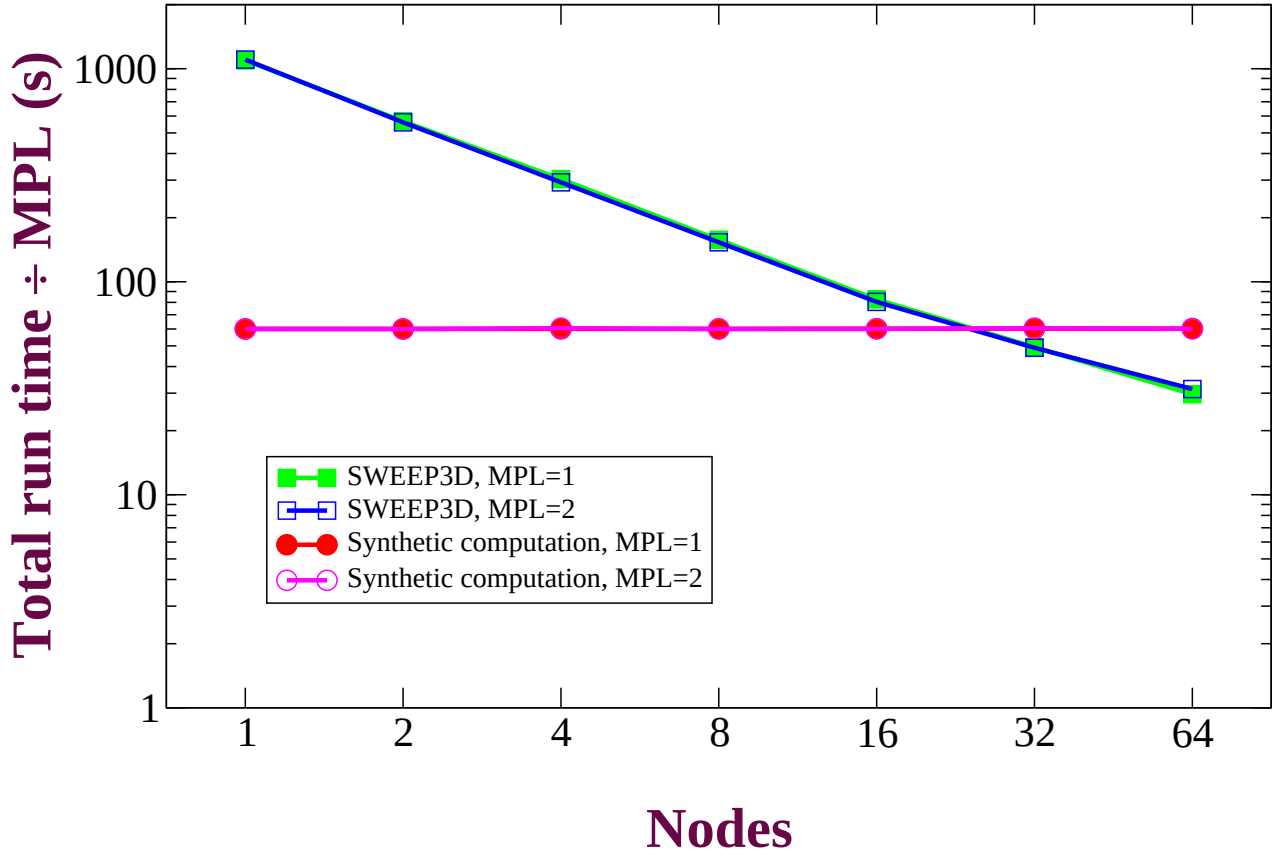


Effect of time quantum with an MPL of 2



Cluster-wide jobs can be scheduled as fast a local process on a desktop OS.

Effect of node scalability



The scheduling quantum is insensitive to the number of nodes

SC2002

A selection of scheduling quanta found in the literature

Resource Manager	Minimal feasible scheduling quantum
RMS	$30,000$ milliseconds on 15 nodes (1.8% slowdown)
SCore-D	100 milliseconds on 64 nodes (2% slowdown)
STORM	2 milliseconds on 64 nodes (no observable slowdown)

Conclusions

- STORM uses an innovative design based on a small set of data-transfer and synchronization mechanisms:
  - XFER-AND-SIGNAL
  - TEST-EVENT
  - COMPARE-AND-WRITE
- STORM is orders of magnitude faster than the best reported results in the literature for both job launching and process scheduling.

Conclusions (continued)

- STORM isolates network specifics
- provides portability
- fast implementation $\rightarrow$ all STORM is fast
- can take advantage of network features (HW multicast, programmable NICs, etc.)

Resources

More information can be found at the following URLs:

Resource management

http://www.c3.lanl.gov/par_arch

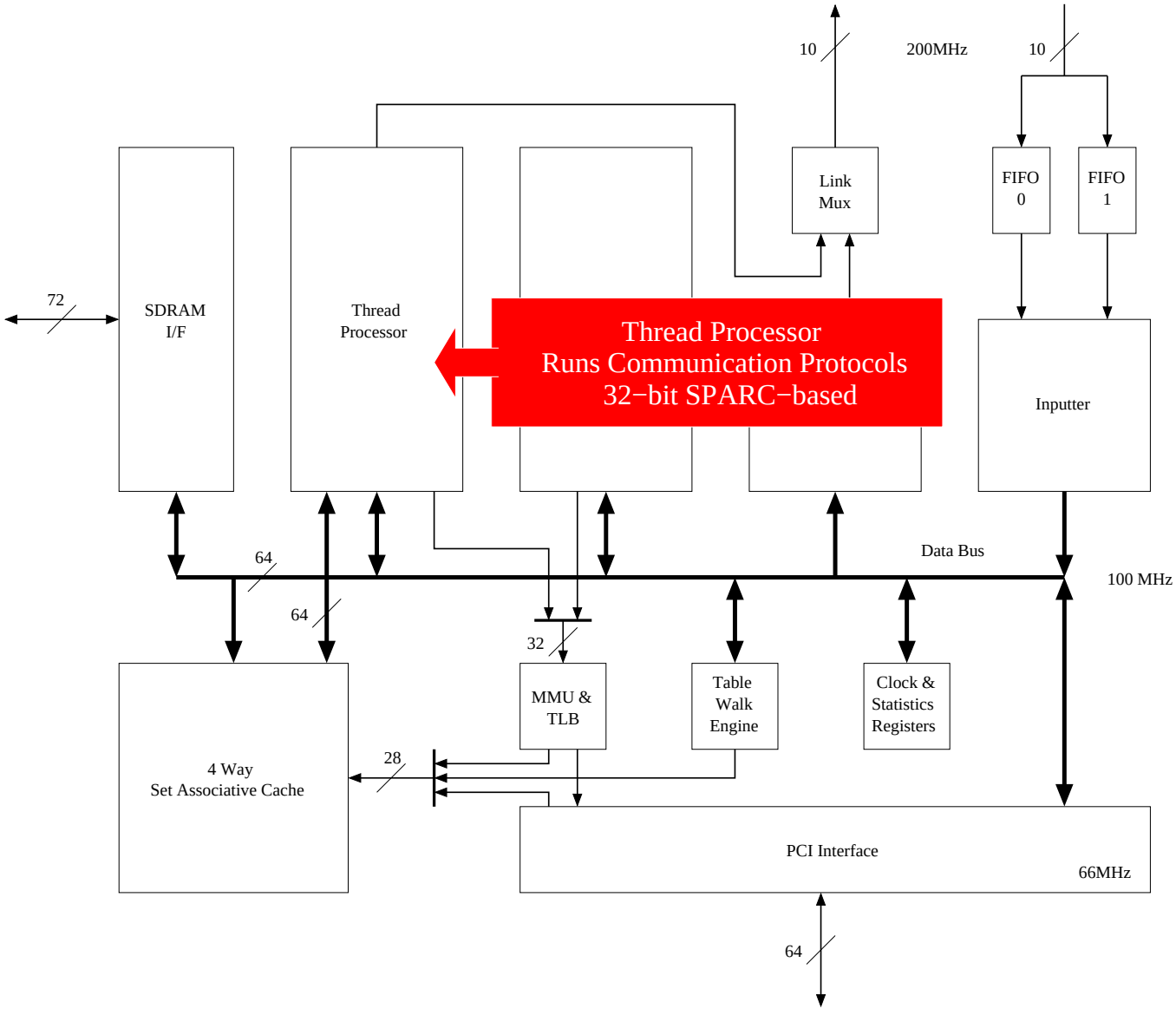
http://www.c3.lanl.gov/~fabrizio/publications.html

Quadrics network

http://www.quadrics.com and

http://www.c3.lanl.gov/~fabrizio/quadrics.html

Quadrics Network: Elan



Quadrics Network: Elan